ERDE: Entropy-Regularized Distillation for Early-exit

Anonymous authors
Paper under double-blind review

Abstract

Although deep neural networks and in particular Convolutional Neural Networks have demonstrated state-of-the-art performance in image classification with relatively high efficiency, they still exhibit high computational costs, often rendering them impractical for real-time and edge applications. Therefore, a multitude of compression techniques have been developed to reduce these costs while maintaining accuracy. In addition, dynamic architectures have been introduced to modulate the level of compression at execution time, which is a desirable property in many resource-limited application scenarios. The proposed method effectively integrates two well-established optimization techniques: early exits and knowledge distillation, where a reduced student early-exit model is trained from a more complex teacher early-exit model. The primary contribution of this research lies in the approach for training the student early-exit model. In comparison to the conventional Knowledge Distillation loss, our approach incorporates a new entropy-based loss for images where the teacher's classification was incorrect. The proposed method optimizes the trade-off between accuracy and efficiency, thereby achieving significant reductions in computational complexity without compromising classification performance. The validity of this approach is substantiated by experimental results on image classification datasets CIFAR10, CIFAR100 and SVHN, which further opens new research perspectives for Knowledge Distillation in other contexts.

1 Introduction

In the field of deep learning, the reduction of computational cost is a matter of significant concern. A multitude of techniques have been identified that adapt Convolutional Neural Network (CNN) architectures, thereby reducing both computational cost and model size. These techniques have proven to be efficient in various aspects of architecture optimization, where the combination of methods is a particularly effective approach to achieving high compression rates when they are complementary, such as a pruning and a quantization method, for example.

For instance, Qi et al. (2021) achieve a compression rate of 50% with an accuracy reduction of only 0.15%-0.37%. Another method proposed a combination early-exit and quantization on CNN (Li et al., 2022) reaching a compression reduction of 50% with an accuracy drop of 1% to 3%. Quantization can also be effectively combined with pruning, e.g. Song Han (2016). Thus, the combination of distinct methods allows to further optimize the trade-off between compression and accuracy.

In this study, we integrate the knowledge distillation and early-exit approaches to obtain distilled dynamic neural networks. That is, at inference time, the distilled network can further reduce its compression rate dynamically by executing only parts of the model depending on external conditions (e.g. battery power) or internal criteria (e.g. the difficulty of the input).

Knowledge distillation (KD) is a training method in which a large reference network, designated as the "teacher", is employed to train a smaller network, referred to as the "student". This approach enables the student network to attain a higher accuracy compared to training without KD. Early-exit (EE), also called multi-exit, methods employ a dynamic compression technique where the network makes multiple intermediate predictions after executing a certain number of layers. The complexity of the model can thus be modulated (automatically or manually) at run-time by exiting earlier from the neural network and thereby eliminating the need for subsequent calculations.

The proposed method named ERDE for Entropy-Regularized Distillation for Early-exit relies on a specific training method that allows us to apply knowledge distillation between two early-exit networks. By effectively combining the two approaches, we are able to cumulate the compression gains of both and, in addition, obtain a run-time configurable model.

In summary, our contributions are the following:

- We present a new compression approach that effectively combines Knowledge Distillation with Early-Exit dynamic neural network architectures.
- We introduce a new distillation method based on a loss function applied at intermediate exits trying to maximize the entropy for those examples where the teacher model is uncertain.

We have tested our approach with different CNNs on three classical image classification benchmarks. Our models obtained a significant reduction in computational complexity, i.e. up to around 10 times lighter than the original models, with little loss in accuracy or even a gain in some cases. In addition, compared to the standard KD training algorithm, applying KD to EE architectures with our proposed entropy-regularization loss improves the average accuracy on all tested datasets for all possible EE thresholds.

2 Related Work

A broad spectrum of model compression strategies has been proposed to alleviate the memory footprint and computational complexity of deep neural networks. These approaches primarily encompass parameter quantization, which reduces numerical precision; network pruning, which removes redundant weights, neurons, or structural components; and Neural Architecture Search, which automates the discovery of compact and efficient model topologies. Collectively, these techniques aim to achieve favorable accuracy–efficiency trade-offs while preserving predictive performance.

2.1 Dynamic Neural Networks

The need for dynamic networks comes from the inability of static ones to adapt the computational graphs or the network parameters. For example, processing complex images requires deep and complex networks that require more computation. However, this type of network will do a large amount of unnecessary computation when processing simpler images.

Many techniques have been developed to adapt the architecture of neural networks at execution time (Han et al., 2021). We can divide these methods into three categories: dynamic depth, dynamic width, and dynamic routing. Each category in turn contains several techniques.

Dynamic depth methods are designed to avoid the redundant computations of deep networks mentioned above. This can be achieved by exiting at shallow exits for simple inputs (early exit) as in Rahmath et al. (2024) or by selectively skipping certain intermediate layers given a particular input (layer skipping) as in Graves (2016) and Wang et al. (2017). Dynamic width methods are more fine-grained than the previous ones, all layers are executed but some units (neurons, channels or branches) are not activated depending on the input. One of the dynamic width methods is Mixture of Experts (MoE) (Jacobs et al., 1991; Eigen et al., 2013). It is a strategy that dynamically selects and uses only a small part of the network (the experts) for processing, based on the given input. Dynamic routing is a key mechanism in modern dynamic neural networks that adapt their computational pathways based on input. It enables the network to optimize its structure by selectively activating specific layers, channels, or paths within architectures like SuperNets (Cha et al., 2022). We can consider early-exiting networks as a special form of SuperNets. Another type of dynamic routing networks are CapsuleNets, where routing between capsules captures hierarchical relationships (Sabour et al., 2017; Hinton et al., 2018).

2.2 Early-Exit Approaches

Early-exit methods add various exits at different layers and are composed of additional classification branches (denoted "exit branches" in the following). The inference is initially performed up to the first exit branch. At this point, a prediction is made with an associated confidence. If the confidence reaches a certain threshold, the inference is stopped and the resulting prediction is returned, otherwise the inference continues until the next exit.

The exit branches can be composed of one or more layersdiffer that can be of different types: either a classic single Fully Connected (FC) layer (Chen et al., 2020; Dai et al., 2020), multiple FC layers (Zhao et al., 2021) or more complex architectures adding one or more convolution layers and pooling layers before the FC layer (Teerapittayanon et al., 2016; Jo et al., 2023). Note that the calculations performed on these exit branches are "lost" if the confidence is below the threshold and thus the exit not used, so it is important to ensure that the exits remain cost effective. The placement of the exit branches may also differ. Some methods add an exit after each layer (Dai et al., 2020), others choose specific layers (Berestizshevsky & Even, 2018; Teerapittayanon et al., 2016) and finally there are more complex techniques like using metrics or gating function to decide where to put the exits (Fang et al., 2020; Li et al., 2023).

The inclusion of additional exits to a classical Deep Neural Network (DNN) entails a different training strategy because the network has several outputs that need to be optimized. Joint training is the most trivial method, in which all branches are trained simultaneously (Teerapittayanon et al., 2016), where the global loss is defined as the sum of all losses obtained at the end of each branch considered by a chosen coefficient. In the branch-wise training strategy, each side branch is trained separately, together with the preceding layers of the backbone DNN (Belilovsky et al., 2019). Separate training consists of treating the exit branches as independent classifiers and training them independently (Chiu et al., 2023). Another training strategy, called two-stage training, is to train the DNN backbone first, then its parameters are frozen and the exit branches are trained separately (Berestizshevsky & Even, 2018). Knowledge Distillation (KD) based training uses the exit branches as student models that learn from the output of the DNN backbone (He et al., 2015; Hinton et al., 2014).

Furthermore, many exit policies have been developed to decide whether to exit or not depending on the branch output. These policies can be divided into two categories: static (rule-based) policies and dynamic (learnable) policies. Static exit policies measure the confidence in the predictions made using metrics such as entropy, maximum softmax or user-defined scoring functions. A single threshold can be applied to all branches, or the threshold can be specified for each branch. These methods are easy to implement and fast, but lack robustness due to their inability to adapt. On the other hand, some learnable exit policies have been proposed, such as exit selection controllers (Dai et al., 2020; Guan et al., 2017), reinforcement learning (Guan et al., 2017; Wang et al., 2017), and soft gating mechanisms (Mullapudi et al., 2018). They have a high training complexity but also a high robustness to diverse input.

2.3 Distillation and Dynamic Neural Networks

One of the commonly used compression methods is called Knowledge Distillation (Hinton et al., 2014). It consists in using a reference model (teacher) to train a smaller network (student), leading to a better accuracy in a shorter time compared to the same student trained without KD. This type of optimisation has been applied to many other techniques to obtain better results. For example, when pruning a network, some work like Chen et al. (2021) shows that using the unpruned network as a teacher helps to obtain better accuracy than classical retraining.

Inspired by Knowledge Distillation, Zhang et al. (2019) have developed self-distillation, which unlike traditional KD, works within a single network. It is a training technique to improve model performance, not a method to compress models. The idea of this method is to divide the network into different sections and distill the knowledge from the deeper sections to the shallower ones. This approach not only improves accuracy but also promotes computational efficiency.

Building on the principles of self-distillation, other researchers have explored its potential in terms of optimisation. For example, ESCEPE (Gourtani & Meratnia, 2023) is an approach based on weight clustering

and self-distillation achieving a high compression ratio of the early-exit network with minimal impact on the accuracy of the intermediate classifiers. The proposed method combines early-exit with self-distillation: a special case of knowledge distillation where the teacher is the network before compression. As apposed to our approach, this work does not explore the possibility of using a different network as a teacher and focuses mainly on pruning a network.

Runtime Neural Pruning (RNP) (Lin et al., 2017) is a framework that prunes networks according to the input and feature maps. Unlike traditional methods that create a fixed pruned model for deployment, RNP preserves the full capacity of the original network, allowing for adaptable pruning. The pruning process is modelled as a Markov Decision Process, and a reinforcement learning agent is trained to determine the optimal pruning policy. Unlike our method, RNP does not explore knowledge distillation at all.

Finally, Xu et al. (2022) propose a multiple-stage knowledge distillation method. It involves the construction of intermediate exits on top of classic networks, facilitating the distillation of knowledge through various points of the network without utilizing the created "early-exit" for dynamic purposes. Additionally, the proposed One-to-One MSKD model exclusively computes the cross-entropy classification loss and the Kullback–Leibler distillation loss. ERDE proffers the utilization of intermediate exits as a strategy to curtail computations and employ an entropy loss on erroneously classified samples to augment network confidence.

A comparison will be made between the proposed architecture and MSDNet (Huang et al., 2017), a multi-scale densely connected architecture designed for anytime prediction. The model maintains parallel feature representations at multiple resolutions and incorporates intermediate classifiers to enable adaptive computation with favorable accuracy–efficiency trade-offs.

3 Method

This section described the methodology utilized and the contribution of the present study. The proposed architecture (illustrated in Figure 1) uses early exits with $n-1$ branches, i.e. there are $n-1$ early exit heads, and the n-th one is the classical exit of the network. The architecture of the EE heads will be defined in the subsequent section. As previously outlined, the premise of early exits is that the example progresses through the network until the first exit. At this point, a primary prediction is conducted, accompanied by the attribution of an uncertainty score c_i to the prediction. This score is defined as the entropy of the softmax vector of the prediction. If the uncertainty score is below a predetermined threshold, the prediction is designated as the network's output, i.e., the final prediction. This predetermined threshold, denoted as θ in $[0; \infty[$, will be consistent for all exits. As the threshold diminishes, a greater proportion of the samples will depart from the lowest exit. Conversely, as the threshold increases, a higher percentage of the samples will exit from the deeper exits. If the score falls below the threshold, the image I will persistently traverse the network until the subsequent exit, a process that is repeated. If the network does not hold sufficient confidence in a prediction at the $n-1$-th exit, the image will terminate at the ultimate exit (corresponding to the conventional end of the network). In this study, entropy was selected as the metric of confidence, a common choice in the literature (Teerapittayanon et al., 2016)).

Algorithm 1 Early-exit inference algorithm.

$i \leftarrow 0$
$h_0 \leftarrow I$
$c_0 \leftarrow \infty$
while $c_i > \theta$ and $i \leq n$ **do**
  $i++$
  $h_i \leftarrow \text{Block}_i(h_{i-1})$
  $y_i \leftarrow \text{EE head}_i(h_i)$
  $c_i \leftarrow \text{entropy}(\text{softmax}(y_i))$
end while
return y_i

We did not explore further EE architectures or policies because we believe this is not part of our contribution. Algorithm 1 formalizes the early-exit inference mechanism.

The training of a multi-exit network necessitates a modification of the conventional training process. This modification is required to ensure the training of all heads and the calculation of the confidence threshold, which is a prerequisite in certain cases. The objective is to consider the various exit losses. The subsequent paragraph provides a detailed definition of these losses.

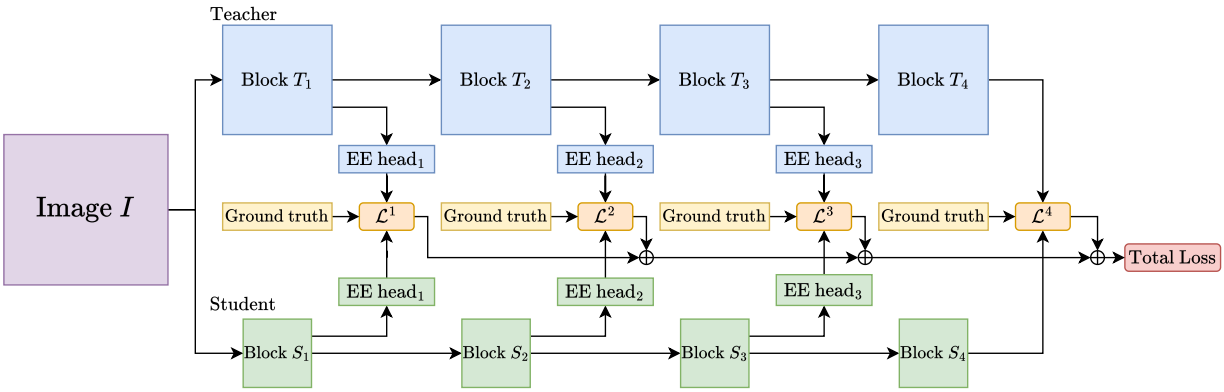


Figure 1: Our ERDE architecture and training approach for 4 teacher and student blocks (in blue and green respectively). The $\mathcal{L}^i$ correspond to the different loss functions ($\mathcal{L}_{\text{CE}}$, $\mathcal{L}_{\text{KD}}$, and $\mathcal{L}_{\text{E}}$) at the i-th exit. At inference, only the compressed student model is used.

Firstly, the following notation must be established: $y_{T_i} = \text{Softmax}(z_{T_i}, T)$, the softmax of the i-th exit of the teacher; $y_{S_i} = \text{Softmax}(z_{S_i}, T)$, the softmax of the i-th exit of the student. Where T is the temperature parameter of the softmax function. Assume, that the number of classes of our problem is K and the number of branches of both, teacher and student, is n.

The Cross-Entropy (CE) loss is one of the classic loss used in deep learning. It is defined as the cross entropy between the ground truth and the student prediction

$$\mathcal{L}^i_{\text{CE}} = -\sum_{k \in K} y_k \log\left(\hat{y}_{S_{i_k}}\right) . \tag{1}$$

This loss compared the class given by the student with the ground truth. $\mathcal{L}^i_{\text{CE}}$ is also used to train the teacher network in the following experiments.

In KD, the classic loss employed is defined as the sum of two losses: the KD loss and the CE loss. The KD loss is expressed as the difference between the student's output and the teacher's output. This loss is designed to encourage the student to make a prediction that mirrors the teacher's prediction. It can be defined through various functions; however, the Kullback-Leibler divergence, which was utilized in the original work of Hinton et al. (2014), was selected for implementation in this study:

$$\mathcal{L}^i_{\text{KL}} = \frac{T^2}{K} \times \sum_{k \in K} \hat{y}_{T_{i_k}} \log\left(\frac{\hat{y}_{T_{i_k}}}{\hat{y}_{S_{i_k}}}\right) . \tag{2}$$

In order to train the student and take into account the early exit, we adapted the classic knowledge distillation method. The primary contribution of the proposed method is the division of the loss between two cases: one for samples that are correctly classified by the teacher and one for samples that are incorrectly classified. In the event that the teacher has successfully classified the sample, we use the combination of $\mathcal{L}_{\text{KL}}$ and $\mathcal{L}_{\text{CE}}$ as usual. In the alternative scenario, the entropy of the output vector is subtracted. Specifically, this entails the subtraction of a quantity $\mathcal{L}_{\text{E}}$ represented as:

$$\mathcal{L}_{\mathrm{E}}^{i} = -\sum_{k \in K} \hat{y}_{S_{i_k}} \log\left(\hat{y}_{S_{i_k}}\right) \ . \tag{3}$$

The idea of this entropy loss is to force the network to be "uncertain" for samples where the teacher failed to accurately predict the class of the image. The maximum of this loss for a sample is achieved when all classes are equally predicted and is minimized when a class is predicted with a probability of 1.

Finally the overall loss can be written as:

$$\begin{aligned}\mathcal{L}_{tot} = &\sum_{i=0}^{n-1} \mathbf{1}_{y=\hat{y}_{T_i}} \left(\omega_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}^{i} + \omega_{\mathrm{CE}}\mathcal{L}_{\mathrm{CE}}^{i}\right) - \mathbf{1}_{y\neq\hat{y}_{T_i}} \omega_{\mathrm{E}}\mathcal{L}_{\mathrm{E}}^{i} \\ &+ \omega_{\mathrm{KL}}\mathcal{L}_{\mathrm{KL}}^{n} + \omega_{\mathrm{CE}}\mathcal{L}_{\mathrm{CE}}^{n} \ .\end{aligned} \tag{4}$$

At the final exit, we employ the conventional KD loss regardless of the accuracy of the teacher prediction. This choice stems from the principle that the final exit is required to produce an output, even for instances where the confidence threshold is not attained. In contrast, the purpose of our entropy loss is to accentuate the uncertainty in intermediate exits, thereby preventing erroneous intermediate predictions.

Table 1: Performance comparison for CIFAR10 for different teacher-student combinations. Reported MMACs and latencies are averages over the whole test dataset.

Approach	Teacher	Student	Accuracy	MACs (M)	rel. MACs (M)	Latency (s)
teacher	ResNet18		0.9153	556.4	100.0%	7.70
student w/o KD		ResNet10	0.9114	254.2	45.7%	6.01
student w KD	ResNet18	ResNet10	0.9275	254.2	45.7%	6.05
ERDE ($\theta = 0$)	ResNet18	ResNet10	**0.9300**	254.2	45.7%	5.86
ERDE ($\theta = 0.4$)	ResNet18	ResNet10	0.9082	**116.8**	**21.0%**	**3.55**
teacher	ResNet34		0.8978	1160	100.0%	11.52
student w/o KD		ResNet10	0.9084	254.2	21.9%	5.99
student w KD	ResNet34	ResNet10	0.9186	254.2	21.9%	6.42
ERDE ($\theta = 0$)	ResNet34	ResNet10	**0.9221**	254.2	21.9%	5.74
ERDE ($\theta = 0.6$)	ResNet34	ResNet10	0.8859	**106.7**	**9.2%**	**3.55**
teacher	ResNet34		0.8978	1160.8	100.0%	11.5185
student w/o KD		ResNet8	0.9009	195.4	16.8%	5.43
student w KD	ResNet34	ResNet8	0.9057	195.4	16.8%	5.99
ERDE ($\theta = 0$)	ResNet34	ResNet8	**0.9060**	195.4	16.8%	5.42
ERDE ($\theta = 0.4$)	ResNet34	ResNet8	0.8856	**103.5**	**8.9%**	**3.92**
teacher	CNXT-b		0.9070	313.6	100.0%	15.98
student w/o KD		CNXT-t	0.8859	91.1	29.0%	10.57
student w KD	CNXT-b	CNXT-t	0.8885	91.1	29.0%	10.96
ERDE ($\theta = 0$)	CNXT-b	CNXT-t	**0.9007**	91.1	29.0%	10.62
ERDE ($\theta = 0.6$)	CNXT-b	CNXT-t	0.8936	**37.9**	**12.1%**	**5.99**
teacher	CNXT-b		0.907	313.6	100.0%	15.978
student w/o KD		ResNet8	0.9009	195.4	62.3%	5.08
student w KD	CNXT-b	ResNet8	0.9041	195.4	62.3%	4.45
ERDE ($\theta = 0$)	CNXT-b	ResNet8	**0.9066**	195.4	62.3%	5.08
ERDE ($\theta = 0.6$)	CNXT-b	ResNet8	0.8883	**109.4**	**34.9%**	**3.60**

4 Experiments

We evaluated our approach on three different standard datasets for image classification: CIFAR10, CIFAR100[1] and SVHN[2]. For SVHN, we did not use the optional extra training data in our experiments. We

[1]https://www.cs.toronto.edu/ kriz/cifar.html
[2]http://ufldl.stanford.edu/housenumbers

tested different student-teacher couples on these datasets including ResNet (He et al., 2016) and ConvNeXT architectures (Liu et al., 2022). We have used 3 different ResNet networks 34, 10, and 8. Where ResNet8 is similar to a ResNet10 but with only 3 blocks (instead of 4 in a ResNet10). We added an exit branch after each block, where the last exit corresponds to the final output of the full model. To minimise the computational overhead, our exit branches are very shallow and operate on the last convolution layer of the preceding block. They contain only a batch normalization layer, a ReLU activation function, a 2x2 average pooling layer, a Dropout with probability 0.5 and a single FC layer that performs the intermediate and final predictions.

Table 2: Performance comparison for CIFAR100 for different teacher-student combinations. Reported MMACs and latencies are averages over the whole test dataset.

Approach	Teacher	Student	Accuracy	MACs (M)	rel. MACs (M)	Latency (s)
teacher	ResNet18		0.6758	559	100.0%	8.68
student w/o KD		ResNet10	0.6521	256.8	45.9%	7.14
student w KD	ResNet18	ResNet10	0.6702	256.8	45.9%	7.29
ERDE ($\theta = 0$)	ResNet18	ResNet10	**0.6741**	256.8	45.9%	7.45
ERDE ($\theta = 0.4$)	ResNet18	ResNet10	**0.6726**	**172.6**	**30.9%**	**5.59**
teacher	ResNet34		0.6654	1163	100.0%	12.91
student w/o KD		ResNet10	0.6471	256.8	22.1%	6.95
student w KD	ResNet34	ResNet10	0.6839	256.8	22.1%	7.01
ERDE ($\theta = 0$)	ResNet34	ResNet10	**0.6880**	256.8	22.1%	6.96
ERDE ($\theta = 0.6$)	ResNet34	ResNet10	**0.6842**	**157.8**	**13.6%**	**5.21**
teacher	ResNet34		0.6654	1163.4	100.0%	12.911
student w/o KD		ResNet8	0.6462	198	17.0%	5.50
student w KD	ResNet34	ResNet8	0.6504	198	17.0%	5.51
ERDE ($\theta = 0$)	ResNet34	ResNet8	**0.6578**	198	17.0%	5.51
ERDE ($\theta = 0.6$)	ResNet34	ResNet8	**0.6536**	**133**	**11.4%**	**4.58**
teacher	CNXT-b		0.7029	314	100.0%	18.71
student w/o KD		CNXT-t	0.7146	91.4	29.1%	13.14
student w KD	CNXT-b	CNXT-t	0.7458	91.4	29.1%	13.89
ERDE ($\theta = 0$)	CNXT-b	CNXT-t	**0.7498**	91.4	29.1%	12.79
ERDE ($\theta = -1.2$)	CNXT-b	CNXT-t	0.7029	**41**	**13.1%**	**8.38**
teacher	CNXT-b		0.7029	314	100.0%	18.7115
student w/o KD		ResNet8	0.6462	198	63.1%	4.55
student w KD	CNXT-b	ResNet8	0.6427	198	63.1%	4.70
ERDE ($\theta = 0$)	CNXT-b	ResNet8	**0.6530**	198	63.1%	4.79
ERDE ($\theta = 1.4$)	CNXT-b	ResNet8	0.6374	**133**	**42.4%**	**3.87**

For each of the datasets, we compared our model trained with our proposed distillation loss (4) to the following baselines: a teacher model without early exits, a student without KD and a student trained with classical KD using the sum of the cross-entropy loss and the distillation loss (KL divergence).

We trained the early-exit models by simultaneously minimizing the loss for all exits, i.e. we simply minimized the sum of all losses without any weighting. The teacher and student models without KD are trained with the CE loss (1) without any (self-)distillation. The student models with classical KD are trained with

$$\mathcal{L}_{\text{KD}} = \sum_{i=0}^{n} \omega_{\text{KL}} \mathcal{L}_{\text{KL}}^{i} + \omega_{\text{CE}} \mathcal{L}_{\text{CE}}^{i} \ , \tag{5}$$

and to evaluate our proposed approach we trained models with the loss in (4).

All models are trained for 300 epochs with a batch size of 64 and a learning rate of 10^{-3} with the Adam algorithm. To avoid overfitting, we applied early stopping with a validation set of 5000 images for CIFAR10 and CIFAR100 and 13256 images for SVHN. For the ConvNeXT models we started with ImageNet pre-trained weights, otherwise the models were not able to converge properly and did overfit too much. Furthermore, data augmentation is performed with random horizontal flips, rotations, translations, crops and random erasing. For all experiments, we set $\omega_{\text{KL}} = 0.25$, $\omega_{\text{CE}} = 0.75$, $\omega_E = 0.005$ and $T = 2$.

Table 3: Performance comparison for SVHN for different teacher-student combinations. Reported MMACs and latencies are averages over the whole test dataset.

Approach	Teacher	Student	Accuracy	MACs (M)	relative MACs	Latency (s)
teacher	ResNet18		0.9623	556.4	100.0%	7.29
student w/o KD		ResNet10	0.9592	254.2	45.7%	5.86
student w KD	ResNet18	ResNet10	0.9637	254.2	45.7%	6.16
ERDE ($\theta = 0$)	ResNet18	ResNet10	**0.9657**	254.2	45.7%	5.91
ERDE ($\theta = 0.4$)	ResNet18	ResNet10	0.9624	**117.3**	**21.1%**	**3.69**
teacher	ResNet34		0.9584	1160	100.0%	10.02
student w/o KD		ResNet10	0.9594	254.2	21.9%	5.59
student w KD	ResNet34	ResNet10	0.9606	254.2	21.9%	6.35
ERDE ($\theta = 0$)	ResNet34	ResNet10	**0.9647**	254.2	21.9%	5.69
ERDE ($\theta = 0.2$)	ResNet34	ResNet10	0.9606	**113.3**	**9.8%**	**3.57**
teacher	ResNet34		0.9584	1160.8	100.0%	10.015
student w/o KD		ResNet8	0.9572	195.4	16.8%	4.87
student w KD	ResNet34	ResNet8	0.9599	195.4	16.8%	4.86
ERDE ($\theta = 0$)	ResNet34	ResNet8	**0.9607**	195.4	16.8%	4.87
ERDE ($\theta = 0.2$)	ResNet34	ResNet8	0.9566	**105.3**	**9.1%**	**3.45**
teacher	CNXT-b		0.9476	313.6	100.0%	15.96
student w/o KD		CNXT-t	0.9468	91.1	29.0%	10.71
student w KD	CNXT-b	CNXT-t	0.9545	91.1	29.0%	10.98
ERDE ($\theta = 0$)	CNXT-b	CNXT-t	**0.9568**	91.1	29.0%	10.69
ERDE ($\theta = 1.4$)	CNXT-b	CNXT-t	**0.9556**	**23.0**	**7.5%**	**4.86**
teacher	CNXT-b		0.9476	313.6	100.0%	15.957
student w/o KD		ResNet8	0.9572	195.4	62.3%	4.867
student w KD	CNXT-b	ResNet8	0.9549	195.4	62.3%	5.010
ERDE ($\theta = 0$)	CNXT-b	ResNet8	**0.9574**	195.4	62.3%	5.089
ERDE ($\theta = 1.6$)	CNXT-b	ResNet8	**0.9557**	**110.1**	**35.1%**	**3.963**

For evaluating the different models, we computed the accuracy and the number of MACs (Multiply-Accumulate operations) and varied the confidence thresholds for the early-exit models. As the number of operations varies for each example, we report the average MACs (per example) over the whole test set. The average latencies are computed with a batch size of 1 on a NVIDIA V100 graphics card including all data loading and transfer overhead.

A part from standard KD, we did not compare to other network compression methods, as they are not directly comparable but they are rather complementary. For example, in pruning approaches, the compression ratio can usually be chosen in a fine-grained manner, whereas in KD we fix a target student architecture in advance.

5 Results

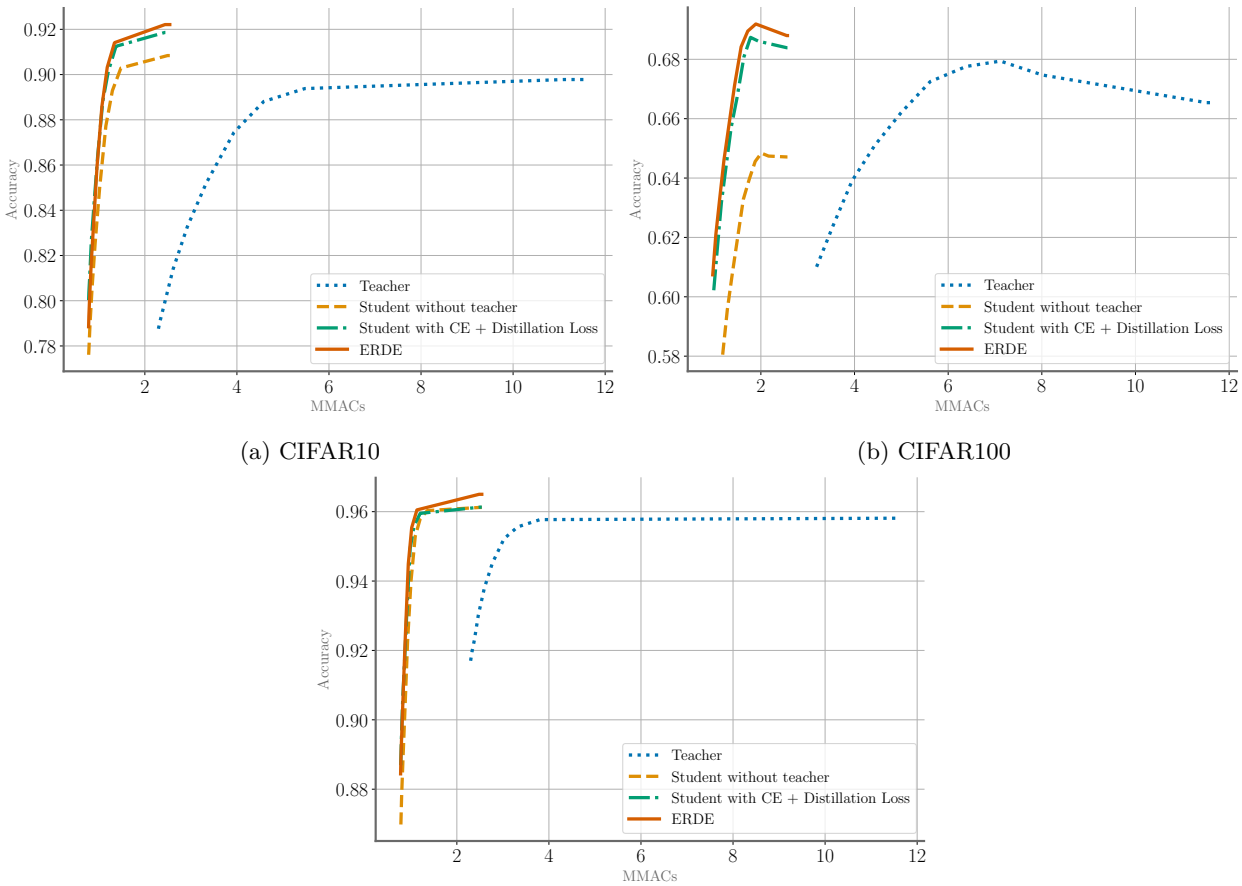
The overall performance results in terms of accuracy, MMACs and latency are shown in Table 1, 2, and 3. The teacher accuracy is usually higher than the one of the student model trained without KD.

However, sometimes this is not the case as the teachers may also tend to overfit. For CIFAR10 and SVHN, the application of knowledge distillation has been demonstrated to yield superior outcomes in comparison to those attained without this approach, as well as those achieved by the teacher. These results are frequently observed in other studies; however, it is noteworthy that the incorporation of our specific entropy loss during the training process has been shown to yield more favorable results in comparison to the conventional distillation loss regardless of the EE threshold and the tested teacher-student architectures. In some cases, our approach increases the accuracy by more than 1% point.

As illustrated in Figure 2a, 2b, and 2c, the early exit threshold has a significant impact on the performance of the models. It is noteworthy that decreasing this threshold can lead to a substantial reduction in the number of MACs without a concomitant decline in accuracy.

Surprisingly, for CIFAR100, it appears that the best accuracy is not obtained when the early exit threshold is maximized. This could be due to some overfitting occurring at the last exit of the model.

As shown in Table 1, 2, and 3, compared to the original teacher models, the models trained with our ERDE approach have only around 10%-40% of MAC operations depending on the chosen architectures with very little decrease in accuracy. In terms of latency, we can achieve a reduction of up to 4 times. In some cases, we may possibly go beyond this by choosing even smaller teacher models.



(a) CIFAR10

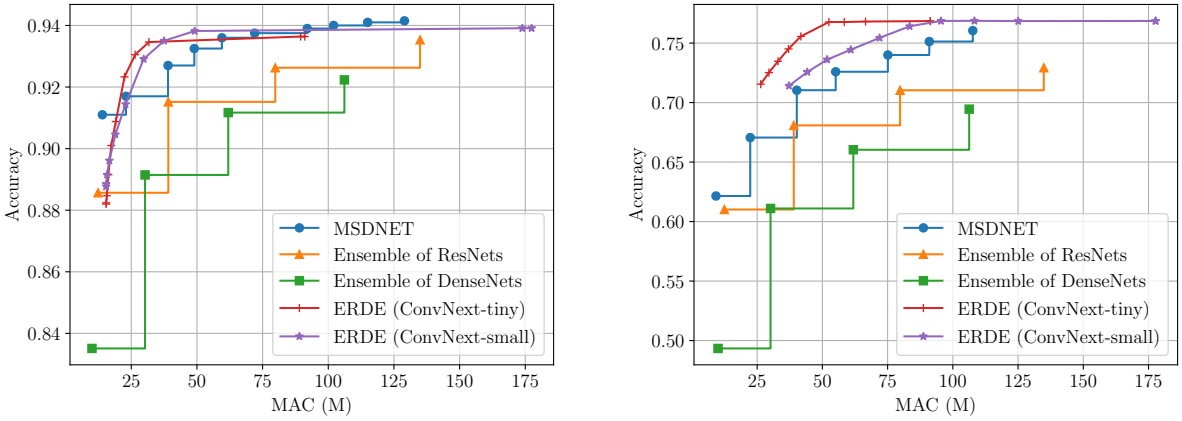
(b) CIFAR100

(c) SVHN

Figure 2: Test accuracy as a function of average MACs for different datasets and training strategies using ResNet34 as teacher and ResNet10 as student.

A comparison was made between our model and MSDNet, a state-of-the-art network. A comparison with different depth ResNet and DenseNet networks is also conducted (see MSDNet (Huang et al., 2017) for more details). The results are displayed in Figure 3. On the CIFAR100 dataset (Figure 3b), the two ERDE networks (utilizing ConvNext-tiny and ConvNext-small as baselines) exhibited superior accuracy-to-MACs trade-off performance in comparison to MSDNet. In the case of CIFAR10 (Figure 3a), however, the outcomes are less obvious. The performance of our models demonstrates a clear superiority in the 15-65 MMAC range in comparison to MSDNET. However, MSDNET has been shown to exhibit a small advantage in terms of achieving optimal results outside this range.

Finally, we have studied the impact of ω_E on a ConvNeXt-tiny network trained using ConvNeXt-base as teacher on CIFAR100. We have tested several values for ω_E from 0 to 5×10^{-2} the resulting accuracy is shown in Figure 4. The best weight for the entropy loss is around 5×10^{-3}. The accuracy is rather stable in the range $[0; 0.02]$ but greater values lead to a significant decrease.



(a) CIFAR10 (b) CIFAR100

Figure 3: Test accuracy as a function of average MACs for different datasets and Networks

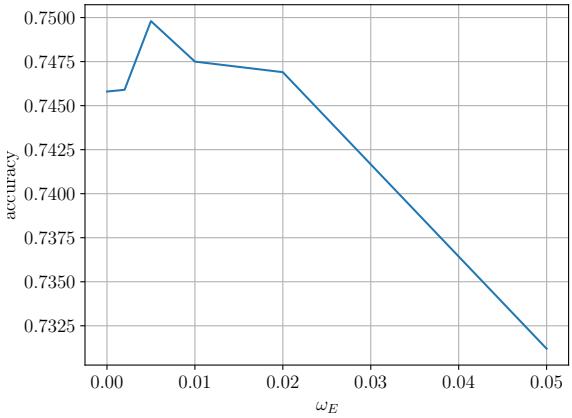


Figure 4: Accuracy of the student for different values of ω_E

6 Conclusion

We have proposed an original method to apply knowledge distillation to an early-exit architecture. This is achieved by using a specific loss and training process. This approach effectively combines knowledge distillation with early exit architectures and thus leads to compact dynamic networks that can control the accuracy-complexity trade-off at run time and leverage both compression methods to reduce the complexity even further. The efficacy of our method has been demonstrated on three distinct datasets on different CNN architectures and different teacher-student combinations. Our method systematically outperforms conventional knowledge distillation and is able to reduce the computational complexity up to around 10 times with negligible decrease in accuracy. It also outperformed MSDNet, ResNet and DenseNet models.

References

Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Shallow learning for deep networks. *unpublished*, 2019.

Konstantin Berestizshevsky and Guy Even. Dynamically sacrificing accuracy for reduced computation: Cascaded inference based on softmax confidence. In *International Conference on Artificial Neural Networks*,

2018.

Stephen Cha, Taehyeon Kim, Hayeon Lee, and Se-Young Yun. Supernet in neural architecture search: A taxonomic survey. *ArXiv*, abs/2204.03916, 2022.

Liyang Chen, Yongquan Chen, Juntong Xi, and Xinyi Le. Knowledge from the original network: restore a better pruned network with knowledge distillation. *Complex & Intelligent Systems*, 8:709 – 718, 2021.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *International conference on machine learning*, pp. 1520–1530. PMLR, 2020.

Ching-Hao Chiu, Hao-Wei Chung, Yu-Jen Chen, Yiyu Shi, and Tsung-Yi Ho. Fair multi-exit framework for facial attribute classification. *unpublished*, abs/2301.02989, 2023.

Xin Dai, Xiangnan Kong, and Tian Guo. Epnet: Learning to exit with flexible multi-branch network. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 235–244, 2020.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. Learning factored representations in a deep mixture of experts. *CoRR*, abs/1312.4314, 2013.

Biyi Fang, Xiao Zeng, Faen Zhang, Hui Xu, and Mi Zhang. Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision. *IEEE/ACM Symposium on Edge Computing*, pp. 84–95, 2020.

Saeed Khalilian Gourtani and Nirvana Meratnia. Escepe: Early-exit network section-wise model compression using self-distillation and weight clustering. *Proceedings of the 6th International Workshop on Edge Systems, Analytics and Networking*, 2023.

Alex Graves. Adaptive computation time for recurrent neural networks. *unpublished*, abs/1603.08983, 2016.

Jiaqi Guan, Yang Liu, Qiang Liu, and Jian Peng. Energy-efficient amortized inference with cascaded deep classifiers. *unpublished*, abs/1710.03368, 2017.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:7436–7456, 2021.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. 2016. URL `https://api.semanticscholar.org/CorpusID:6447277`.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning Workshop*, 2014.

Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *ArXiv*, abs/1703.09844, 2017. URL `https://api.semanticscholar.org/CorpusID:1367596`.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

Joongho Jo, Geonho Kim, Seungtae Kim, and Jongsun Park. Locoexnet: Low-cost early exit network for energy efficient cnn accelerator design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42:4909–4921, 2023.

Xiangjie Li, Chen Lou, Zhengping Zhu, Yuchi Chen, Yingtao Shen, Yehan Ma, and An Zou. Predictive exit: Prediction of fine-grained early exits for computation- and energy-efficient inference. *ArXiv*, abs/2206.04685, 2022.

Xiangjie Li, Yingtao Shen, An Zou, and Yehan Ma. Eenet: Energy efficient neural networks with run-time power management. *60th ACM/IEEE Design Automation Conference*, pp. 1–6, 2023.

Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. In *Neural Information Processing Systems*, 2017.

Zhuang Liu, Hanzi Mao, Chaozheng Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11966–11976, 2022. URL `https://api.semanticscholar.org/CorpusID:245837420`.

Ravi Teja Mullapudi, William R. Mark, Noam M. Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8080–8089, 2018.

Q. Qi, Yanyang Lu, Jiashi Li, Jingyu Wang, Haifeng Sun, and Jianxin Liao. Learning low resource consumption cnn through pruning and quantization. *IEEE Transactions on Emerging Topics in Computing*, 10:886–903, 2021.

Haseena Rahmath, Vishal Srivastava, Kuldeep Chaurasia, Roberto Gonçalves Pacheco, and Rodrigo De Souza Couto. Early-exit deep neural network - a comprehensive survey. *ACM Comput. Surv.*, 57: 75:1–75:37, 2024.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

William J. Dally Song Han, Huizi Mao. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

Surat Teerapittayanon, Bradley McDanel, and H. T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. *23rd International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

Xin Wang, Fisher Yu, Zi-Yi Dou, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. *unpublished*, abs/1711.09485, 2017.

Chuanyun Xu, Nanlan Bai, Wenjian Gao, Tian Li, Mengwei Li, Gang Li, and Yang Zhang. Multiple-stage knowledge distillation. *Applied Sciences*, 2022. URL `https://api.semanticscholar.org/CorpusID:252481152`.

Linfeng Zhang, Jiebo Song, Anni Gao, Jingwei Chen, Chenglong Bao, and Kaisheng Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. *IEEE/CVF International Conference on Computer Vision*, pp. 3712–3721, 2019.

Zhihe Zhao, Kai Wang, Neiwen Ling, and Guoliang Xing. Edgeml: An automl framework for real-time deep learning on the edge. In *Proceedings of the international conference on internet-of-things design and implementation*, pp. 133–144, 2021.